Overview

In this Assignment you will practice with text classification. Given a training collection of classified documents, you will build a classifier that you will use to classify another (larger) set of documents. You will compare the performances of Naive Bayesian and Rocchio classifiers, and understand how selecting different features affects their performance.

Please read carefully the whole document before starting coding. If you are in doubt, ask the TAs.

1 Specification

You will code two programs. The first one, `vecrep`, takes as input a collection of Wikipedia pages and a list of term features, and outputs the representation of each page as a vector in the features space. The second program, `classify`, will train a text classifier on a provided set of classified documents and use it to label the pages in a second (larger) set of pages.

1.1 Representing the documents as vectors in the feature space

In Project Part 1 and Part 2, you obtained a representation of the documents as streams of terms. (Please refer to Section 1.1.1 of the specification for Part 1 for the details.)

When dealing with textual information, classifying based on the full vocabulary is inefficient and might even negatively affect the classification quality. Therefore, your first program `vecrep` will take care of producing a vector representation of the documents: given the collection, you should obtain the representation of each document as a stream of terms \( d = \langle t_0, t_1, \ldots \rangle \) (note that a term might be repeated several times), you will project this document on the space generated by the given list of terms (features) \( F = \langle f_0, f_1, \ldots, f_{m-1} \rangle \). Therefore, each document \( d \) will be represented as an \( m \)-dimensional vector, where the \( i \)-th component will be the number of occurrences of feature \( f_i \) in document \( d \). Since these vectors are very sparse, you will output them as a list of `feature_index : value` pairs, as described in the next section.

Please refer to Figure 1 in Appendix for an example.

1.1.1 Input/Output

The format of the list of stop words and of the collection of pages are as in the previous assignments. The list of features is given as a text file containing a term per line. (As usual, we assume that the first feature has index zero.)
The output file should have a line for each document, in order of docID, and each line should be in the following format:

\[ d \text{ sum}_d f_{i:occ_i} \ldots \]

where:

- \( d \) is the docID of the document;
- \( f_i \)'s are the term features appearing in document \( d \), in ascending order of featureID;
- \( occ_i \) is the number of occurrences of \( f_i \) in document \( d \);
- \( \text{sum}_d \) is the sum of the squares of the \( occ_i \)'s.

For example, if the documents in your collection correspond to the following three vectors:

\[
\begin{align*}
\text{d}_0 &= < 0, 1, 1, 2, 1 > \\
\text{d}_1 &= < 2, 2, 0, 0, 0 > \\
\text{d}_5 &= < 0, 0, 0, 0, 0 >
\end{align*}
\]

your output file will be:

\[
\begin{align*}
0 & 7 1:1 2:1 3:2 4:1 \\
1 & 8 0:2 1:2 \\
5 & 0
\end{align*}
\]

Note that you do not need to output the featureIDs not appearing in a document and that it might happen that a document is mapped to the null vector. Please refer to Figure 1 in Appendix for a full example.

You will submit a bash script `vecrep.sh`, suitably invoking your `vecrep` program. Your bash script should accept four parameters, representing names of files:

- the list of stop words;
- the collection of pages;
- the list of features;
- the file name of the vector representation to be built.

To run your program, we will invoke something like this:

```
$ ./vecrep.sh stopWords.dat fullCollection.dat features.dat vecrep.dat
```
1.2 Training and using the classifiers

Using the vector representation explained in the previous section, you will train the classifier using a given training set of (docID, categoryID) pairs. Your classify program will implement two classification methods, namely Multinomial Naive Bayes (MNB) and Rocchio classifiers, selectable by the user at runtime. Since you will re-use this code for the Final Assignment, please write it in a modular way!

For the Multinomial Naive Bayes model, please refer to Section 13.2 of the textbook. You are required to implement the two algorithms of Figure 13.2 (TRAINMULTINOMIALNB and APPLYMULTINOMIALNB). Please use both Laplace smoothing and log-probabilities to avoid numerical underflow. Note that the vocabulary \( V \), in the notation of Figure 13.2, is the provided list of term features.

The Rocchio classifier is described in Section 14.2 of the textbook. You are required to implement the two algorithms of Figure 14.4 (TRAINROCCIO and APPLYROCCIO). Note that you will need to normalize the vectors in order to correctly compute the centroids (Equation (14.1)).

Please observe that some documents that were used in the training set might get a different label during classification.

1.2.1 Input/Output

The training set will be a plain text file, with a line for each (docID, categoryID) pair. The format is the following:

```
0 1
1 6
2 0
5 6
```

meaning that document 0 belongs to category 1, document 1 to category 6, and so on. Note that the docIDs and the categoryIDs start with index zero.

After training your classifier, you will use it to determine the labels for a given list of documents, identified by their docIDs. This list will be provided as a plain text file, with a docID in each line. Your classifier should output, for each of the requested docIDs, the predicted label. The format of the output file is the same as for the training set (i.e, docID categoryID, one per line).

You will submit a bash script classify.sh, suitably invoking your classify program. Your bash script should accept six parameters:

- the classification method to be used (-mnb for the MNB or -r for Rocchio);
- the list of features;
- the vector representation of the pages;
- the file name of the training set;
- the file name containing the list of documents to be classified;
- the file name of the classification results to be generated.
To run your program, we will invoke something like this:

```
$ ./classify.sh -mnb features.dat vecrep.dat training.dat docs.dat resultsMNB.dat
$ ./classify.sh -r features.dat vecrep.dat training.dat docs.dat resultsRocchio.dat
```

## 2 Data

For this assignment you will need the following files located in `/course/cs158/data/classif/`:

- `fullCollection.dat`: the collection of documents (warning: 325 MB!);
- `categories.dat`: list of the categories;
- `features[1-2].dat`: two sets of term features;
- `training[1-2].dat`: two training sets of classified documents;
- `test.dat`: test set of classified documents;
- `stopWords.dat`: the list of stop words;
- `vecrep.sh`: sample Bash script to invoke your `vecrep` program;
- `classify.sh`: sample Bash script to invoke your `classify` program;
- `readme.txt`: a plain text file you should fill in with your information;
- `results.txt`: a plain text file you should fill in with the classification results.

## 3 Report

Please include a report in PDF format answering the questions. *Please make your answers as concise as possible. Be concrete, precise and concise. Please do not answer questions that we do not ask! If you are not sure, ask the TAs.*

1. Build the vector representation of the collection of documents for each of the provided features lists.

2. Run both the Naive Bayesian and Rocchio classifiers using `training1.dat` and each of the provided features sets. First, re-classify the documents in the training set and compute the classification error (% of misclassified documents). Then, classify the documents in `test.dat` and compute the classification error on this test set. (1 table) Report the results obtained using each of the features lists for both classifiers. Be sure to fill out the appropriate information in `results.txt`.

3. Look at the results obtained in the previous two points, inspect the provided features lists and explain their impact on the classification accuracy.

4. Repeat Point 2, using `training2.dat`, again including the information in `results.txt`. 
5. Inspect training2.dat, then look at the results obtained in the previous point: can you explain them?

6. Build your own list of features, named featuresADV.dat, with the goal of using it for classifying the document collection. Describe how you built it and the rationale behind your choices.

7. Repeat Point 2, using training1.dat and featuresADV.dat. (1 table) Compare your results with those obtained with the provided features lists: did you improve the previous classification results? Report the confusion matrix for both MNB and Rocchio.

4 Collaboration Policy

You can work on the Course Project in pairs, with the same team-mate with whom you worked on the previous assignments. Only one submission per team is required. Please use http://groups.google.com/group/brown-cs158-spring2011 or email cs158tas at cs dot brown dot edu if you have any questions.

5 Submission

Please copy all the files mentioned below into a separate directory, cd there, and run the handin script from that directory. Be aware that since the handin script copies recursively all the files from the directory it is executed from, if you run the script from, say, your home directory, it will handin all your files and directories! The directory from where you run the handin script should contain only:

- Any source code you wrote and any external module (e.g., .py/.pyc) you need to run your code.
- The two Bash scripts vecrep.sh and classify.sh, suitably modified to call your programs.
- The plain text file readme.txt, filled in with the required information. (Please do not edit the structure of this file (i.e. FULL_NAME, etc.), just substitute our data with yours!)
- Your own list of features, named featuresADV.dat.
- The plain text file results.txt, filled in with the required information. (Please do not edit the structure of this file.)
- Your report, in PDF format, named report.pdf.

Please submit only the above files (do not submit the collection or other unnecessary large files!) using the following command:

```
$ /course/cs158/bin/cs158-handin classif
```

We suggest you to create an empty directory, copy all the above files there, and then create a copy of the entire directory. Move to the latter and try to execute your bash scripts, in order to check that you do not forget to include files necessary to the execution of your programs. If your programs run smoothly, then you can submit your files from the “clean” directory.
6 Evaluation

We will run your programs on a CS machine, verifying that your programs produce the reported results.

It is mandatory that your programs adhere to the I/O interface described in this specification. In particular, check for unwanted prompts, debugging messages, newlines that you might have forgotten in your code. It is your responsibility to make sure that your code runs on the department machines and satisfies naming and usage criteria described in this document. We will only evaluate submissions that respect the I/O specification, requiring you to fix your code, until it complies with the interface described in this document. Moreover, we expect that your programs will run without any syntax or runtime errors.

Your grade will depend on:

- The correctness of your programs. (50%)
- The quality of your report. (50%)

Have fun!

Appendix

Please see the next page for an example of vector representation of a document.
<page>
  <id>0</id>
  <title>Description of a Page</title>
  <text>
  Page content, in Wikipedia format.
  It might span multiple lines,
  with blank lines
  and [[internal links]],
  punctuation, strange
  characters !@#$%^&*,
  numbers like 2011, etc.
  </text>
</page>

(a) Original page.

descript page page content wikipedia format
might span multipl line blank line intern link
punctuat strang charact number like 2011 etc

(b) The final stream of terms, after stemming.

algorithm
charact
content
line
link
note
punctuat
wikipedia

d = <0, 1, 1, 2, 1, 0, 1, 1>

(d) Top: Vector representation of the page, with respect to
the features list on the left. Bottom: sparse output format.

c) List of (term) features.

Figure 1: Obtaining the sparse vector representation of a page.